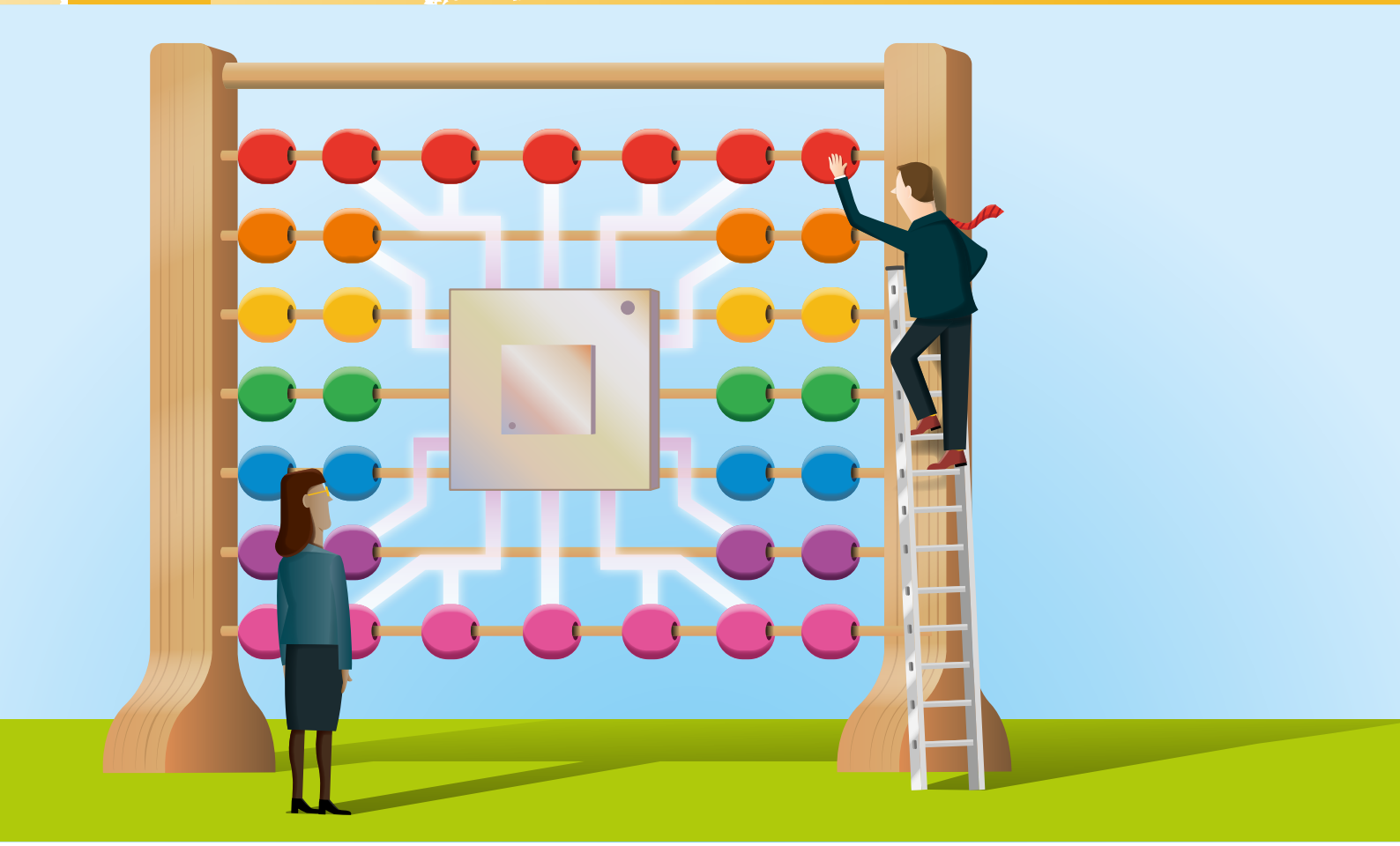

COMPUTATIONAL THINKING AND MATHEMATICAL REASONING

Introducing our maths-themed series of features, **Miles Berry** considers the link between computer science and mathematics...

For me personally, mathematics and computer science have always been closely linked. I was first taught BASIC during secondary school maths lessons. My 'O' Level programming project involved calculating lines of best fit and correlation coefficients. Studying mathematics at university, I learnt ML and discrete mathematics through sitting in on CompSci lectures and classes, undertook the optional programming projects, and got summer jobs developing computer models and overhauling Fortran code.

Starting out as a secondary maths teacher, I was keen to integrate technology into my mathematics lessons: I got some of my lessons timetabled for the computer lab, and thus had a good excuse for teaching a little programming for problem-solving in maths. I first learned about Linux and Python in a hands-on maths professional development workshop, ran by the NRICH maths enrichment programme in Cambridge.

Whilst I know there are lots of subjects to which we can connect computing, I remain convinced that there are some particularly powerful synergies that we can exploit when we link CS with maths. There are ample ways to draw on coding in maths lessons – especially so if you've a language that supports functions (such as Snap!, Python, Pyret, as used by Bootstrap World, or even Haskell), and a certain amount of mathematical content that we need to cover in CS teaching (for example, binary and other number bases, Boolean logic, and integer arithmetic), What I'd like to explore here are the strong parallels between computational thinking and mathematical reasoning.

Computations

Before Turing's time, the word 'computer' was actually a job title – folks spent their days performing calculations by hand, following the (sometimes complex) instructions their bosses gave them. I do worry that a lot of school maths seems to be still caught up in this mindset: much of the time seems to be spent getting children to perform routine computations, be they in arithmetic, algebra, or calculus, rather than thinking creatively about solving interesting problems.

At the very foundation of computer science as an academic discipline, Turing sought to establish what it was to do computation: his insight was that this was essentially manipulating symbols on paper according to a set of rules, and that, whilst such computations could be performed by people, they could also be done by machines following the same rules.

> "THE CODE OUR STUDENTS WRITE DRAWS ON SUB-SYSTEMS AND FITS IN TO SUPER-SYSTEMS

Over 80 years on, arithmetic, algebra, calculus, geometry, statistics and the rest of maths gets done by computers apart, of course, from in school! For school mathematics to better reflect real world mathematics, shouldn't children spend less of their time doing the work that the machines can do, and more of their time thinking about what rules the machines should follow? In short, shouldn't more of a maths lesson be spent doing computational thinking, and rather less time doing computation?

George Pólya's 1945 book, *How to Solve It* ('A system of thinking which can help you solve any problem'), identified four distinct individual stages to mathematical problem solving:

- Understanding the problem
- Devising a plan
- Carrying out the plan
- Looking back

There are quite clear parallels here with our 'computational thinking': abstraction, decomposition, and generalisation all help with understanding the problem; devising a plan draws directly on algorithmic reasoning; and looking back mirrors evaluation. Carrying out the plan was calculation for Pólya; for us, it's the coding. This approach is a very general one, as Pólya demonstrates in his book, and has stood the test of time well. I'd feel just as confident recommending this as a framework for linking computational thinking and coding in the computer science classroom today as I did for mathematical problem-solving in the maths classroom.

Amongst the many brilliant insights that Seymour Papert shared in *Mindstorms* was that pupils' mathematical reasoning was helped through their learning to code: for Papert, code connected pupils with deep ideas in mathematics at an almost visceral level, although, to be fair, he offered little by way of hard data to support this view. This connection wasn't so much about applying the formulae for the exterior angles of a polygon to draw shapes in Logo, as it was about children discovering the rule through putting oneself in the place of the turtle.

Similarly, I suspect many pupils now form a mental model of four quadrant coordinates through creative play in Scratch way before this gets taught to them in their maths lessons. I could imagine older pupils developing a feel for complex ideas in spatial geometry or statistics through playful, creative engagement with 3D modelling and big(ish) data.

Reasoning

These connections between mathematical reasoning and computational thinking are implicit in the English national curriculum. For example, in computing, 11-14 year olds are taught to:

"design, use, and evaluate computational abstractions that model the state and behaviour of real-world problems and physical systems"

Whereas, over in the maths curriculum, we read that pupils,

"can solve problems by applying their mathematics to a variety of routine and non-routine problems with increasing sophistication, including breaking down problems into a series of simpler steps and persevering in seeking solutions.

Abstraction lies at the heart of both computational thinking and mathematical reasoning – in both we capture or model something of the real world – the rules and relationships, the states and behaviours, that lie at the heart of the problems we're solving or the systems we're studying. However, there's nonetheless some difference between how the two domains treat abstraction: in maths, abstraction typically involves getting rid of irrelevant details. Yet in computing, abstraction works in two differing directions: the code our students write draws on sub-systems and fits in to super-systems. Jeanette Wing argued that computational abstraction was more general than its mathematical equivalent, and more concrete, as they are implemented to run on actual, physical hardware.

I'm hopeful that, as our pupils become more fluent with code, they start to look at the problems we set in maths lessons and homework from a coder's perspective. Maybe we might even get to the point where their teachers would accept a program (or an algorithms) to find the solution, rather than an insistence that all working be shown. (HW)

MILES BERRY

Miles (@mberry) is Principal Lecturer in Computing Education at the University of Roehampton. He's a member of the Raspberry Pi Foundation, and serves on the boards of CAS and CSTA.